Ramesh

Q: A Developer's Guide

Version 0.1

June 12, 2017

Contents

Q: A Developer's Guide

1 Introduction

The aim of this document is to provide a step-by-step guide to being a library developeri, working on the internals of Q. We will also discuss how packaging and installation are done and how to operate as a Q developer. A Q developer need have **no** idea about the internals of Q — it is just another Lua package. Of course, understanding the internals, to some extent, allows for more efficient usage of the Q primitives.

2 Getting Started

2.1 Environment Variables

So, you want to modify the guts of Q? Here's a step by step guide.

1. Say, you are in `/home/subramon/WORK/`
2. `git clone https://github.com/NerdWalletOSS/Q.git`
3. Set the following environment variables using `source setup.sh -f`. Note that this is just a convenience. If you want, you can set them yourself but then the onus is on you to get things right. These are
   (a) `Q_CFLAGS` — these will be used as flags to gcc when creating `.o` files
   (b) `Q_LINK_FLAGS` — these will be used as flags to gcc when creating `.so` files
   (c) `Q_ROOT` — This is where artifacts created by the build provess will be stored. As of now, they are
      i. `Q_ROOT/lib/` — contains `libq_core.so`
      ii. `Q_ROOT/include/` — contains `q_core.h`

iii. `Q_ROOT/tmpl/` — contains templates, used for dynamic code generation

(d) `Q_DATA_DIR` — This is where data files will be stored. Think of this as a tablespace and keep a separate one for each project you are working

(e) `Q_METADATA_DIR` — This is where meta data files will be stored. Think of this as a tablespace and keep a separate one for each project you are working on. Default will be `Q_ROOT/meta/`

(f) `LD_LIBRARY_PATH` Make sure that this includes `Q_ROOT/lib/` This is where `libq_core.so` will be created

(g) `LUA_PATH`, Section 2.3

2.1.1 Consequences

There are some important consequences of the above.

1. **Do not set** these environment variables in any of your scripts.
2. **Do not use** `Q_ROOT` anywhere except in `Q/UTILS/build`

2.2 Building

C programs are used to augment Lua in two important ways

1. to help with code generation and to perform some functionality that could not be done easily (or in a performant manner) in Lua. Examples of these are text converters like `txt_to_I8` or `get_cell`
2. the computational workhorse. This is where the heavy lifting happens.

You will note a bit of a circular dependency. We need C code to create C code. This is broken in one of two ways

1. Execute `Q/UTILS/mk_core_so.lua` This creates the following files
   (a) `Q_ROOT/include/q_core.h` — which is used for `ffi.cdef()`
   (b) `Q_ROOT/lib/q_core.so` — which is used for `ffi.load()`

   You are have the C functionality needed for code generation
2. Within `Q/UTILS/build/`, do `make clean; make`

2.3 Masquerading as a Q developer

From time to time, you will need to pretend to be a Q developer so that you can test your code. To enable this to happen without re-installing Q, you set `LUA_PATH` as below. Note the double semi-colon at the end. That is needed. Srinath to fill in the gaps **TO BE COMPLETED**

```
\verb+/home/subramon/WORK/?.lua;;+
```